Apple Assembly Line

$1.80

Volume 6 -- Issue 1 October, 1985

In This Issue...

Book, Books, Books

Inside this issue you will find a review of Jules Gilder's new book on intermediate-level Apple assembly language programming, and the details on those long-awaited Addison-Wesley editions of Apple's Technical Manuals. We're now offering these items for sale, and the details are in our ad.

The latest word from Prentice-Hall is that David Eyes' "Programming the 65816" will be shipped on October 29, so we may actually have copies by the time you read this. Bob will have a full review next month, and we are beginning to get orders already. The list price is expected to be $22.95. If that holds, our price will be $21.00 + postage.

A Rumor Regarding the Next Apple II

We have heard from two sources now a rumor that Apple does not plan to use the 65816 in its next Apple II. Nor the 65802, nor the 65C02. Instead, we heard, they will use a custom version of the 68000 family with 65C02 emulation capability. I think that I hope that the rumor is groundless, but I'll keep my ear to the ground anyway.

ProDOS Snooper.............................Bob Sander-Cederlof

This past week I have been working on a project which involved creating a new device driver for a disk-like device. In the process of debugging my driver, I had to write a "snooper" program.

By "snooper", I mean a program which will make a list of all calls to the driver, recording the origin of the call and the parameters of the call.

ProDOS keeps a table of the addresses of the device drivers assigned to each slot and drive between $BF10 and $BF2F. There are two bytes for each slot and drive. $BF10-1F is for drive 1, and $BF20-2F is for drive 2. For example, the address of the device driver for slot 6 drive 1 is at $BF1C,1D. (Normally this address is $D000.)

I have a Sider drive in slot 7. The device driver address for the Sider is $C753, and is kept at $BF1E,1F and $BF2E,2F.

By patching the device driver address to point to my own code, I can get control whenever ProDOS tries to read or write or whatever. If I save and restore all the registers, and jump to the REAL device driver after I am finished, ProDOS will never be the wiser. But I will!

While my program has control, I can capture all the information I am interested in. Unfortunately I cannot print it out at this time, because if I try to ProDOS will get stuck in a loop. Instead I will save the data in a buffer so I can look at it later.

The program which follows has three distinct parts. Lines 1140-1290 are an installation and removal tool. If the program has just been BLOADed or LOADed and ASMed, running INSTALL.SNOOPER will (you guessed it!) install the snooper. The actual device driver address for the slot (which you specified in line 1060 before assembling the program) will be saved in my two-byte variable DRIVER. The previous contents of DRIVER, which is the address of my snoop routine, will be copied into ProDOS's table. The value of DRIVES, which you specified before assembling the program at line 1070, will determine whether SNOOPER is connected to drive 2 or not. It will always be connected to drive 1.

If SNOOPER has already been installed, running INSTALL.SNOOPER will reverse the installation process, returning ProDOS to its original state. INSTALL.SNOOPER also resets the buffer I use to keep the captured information. To make it easy to run INSTALL.SNOOPER, I put a JMP to it at $300. After assembly you can type "$300G" to install the snooper, and type the same again to dis-install it.

The JMP at $303 (line 1120) goes to the display program. After SNOOPER has been installed, all disk accesses on the installed slot will cause information to be accumulated in BUFFER. Typing "$303G" will cause the contents of BUFFER to be

```
S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
ProDOS Upgrade Kit for Version 2.0 DOS owners...........................$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners..........................$20
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM.........................................................$30
Source Code for DISASM.....................................additional $30
S-C Word Processor (with complete source code)..........................$50
DP18 Source and Object..................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk.................................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
                8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
 Each disk contains the source code from three issues of AAL,
 saving you lots of typing and testing.
 The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)......................... package of 20 for $32 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers.............................  6 cents each

65802 Microprocessor (Western Design Center)..................($95)   $50 *
quikLoader EPROM System (SCRG)...............................($179)  $170 *
PROmGRAMER (SCRG).........................................($149.50)  $140 *
Switch-a-Slot (SCRG).........................................($190)  $175 *
Extend-a-Slot (SCRG)..........................................($35)   $32 *

"Programming the 65816", Eyes...............................($22.95)   $21 *
"Apple //e Reference Manual", Apple Computer................($24.95)   $23 *
"Apple //c Reference Manual", Apple Computer................($24.95)   $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95)   $27 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)   $18 *
"Apple ProDOS:  Advanced Features for Programmers", Little..($17.95)   $17 *
"Inside the Apple //c", Little..............................($19.95)   $18 *
"Inside the Apple //e", Little..............................($19.95)   $18 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)   $18 *
"Apple ][ Circuit Description", Gayler......................($22.95)   $21 *
"Understanding the Apple II", Sather........................($22.95)   $21 *
"Understanding the Apple //e", Sather.......................($24.95)   $23 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)   $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)   $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)   $20 *
"Beneath Apple DOS", Worth & Lechner........................($19.95)   $18 *
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)   $18 *
"6502 Subroutines", Leventhal...............................($18.95)   $18 *
"Real Time Programming -- Neglected Topics", Foster..........($9.95)    $9 *
"Microcomputer Graphics", Myers.............................($12.95)   $12 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95)   $16 *
"Assembly Lines -- the Book", Wagner .......................($19.95)   $18 *
"AppleVisions", Bishop & Grossberger........................($39.95)   $36 *

              * On these items add $2.00 for the first item and
                $.75 for each additional item for US shipping.
                Foreign customers inquire for postage needed.
          Texas residents please add 6 1/8 % sales tax to all orders.
```

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
***   (214) 324-2050   ***
*** We accept Master Card, VISA and American Express ***

displayed in an easy-to-read format.

I set up SNOOPER to capture eight bytes of information each time it is activated. You might decide to save more or less. I save the return address from the stack, to get some idea of which routine inside ProDOS is trying to access the disk. I also save the six bytes at $42-47, which are the calling parameters for the device driver. Page 6-8 of Beneath Apple ProDOS describes these parameters; you can also find out about them in Apple's ProDOS Technical Reference Manual and in Gary Little's "Apple ProDOS--Advanced Features".

$42 contains the command code: 00=status, 01=read, 02=write, and 03=format. $43 contains the unit number, in the format DSSS0000 (where SSS=slot and D=0 for drive 1, D=1 for drive 2). $44-45 contain the address of the memory buffer, lo-byte first; the buffer is 512 bytes long. $46-47 contain the block number to be read or written.

My DISPLAY program displays each group of eight bytes on a separate line, in the following format:

```
     hhll:cc.uu.buff.blok
```

where hhll is the return address from the stack, hi-byte first; cc is the command code; uu is the unit number; buff is the buffer address, hi-byte first; blok is the block number, hi-byte first.

If you get into figuring out more of what ProDOS is doing, you might want to save more information from the stack. You can look behind the immediate return address to get more return addresses and other data which have been saved on the stack before calling the device driver

A word of explanation about lines 1040, 1360, 1370, 1490, and 1500. Line 1040 tells the S-C Macro Assembler that it is OK to assemble opcodes legal in the 65C02. The PHX, PHY, PLX and PLY opcodes are in the 65C02, 65802, and 65816; however, they are not in the 6502. If you have only the 6502 in your Apple, you will need to substitute the longer code shown in the comments. Leave out line 1040, and use the following:

```
       1360    TYA
       1365    PHA
       1370    TXA
       1375    PHA
       .
       .
       .
       1490    PLA
       1495    TAX
       1500    PLA
       1505    TAY
```

In the process of "snooping" I was able to debug my new device drivers for the project I was developing. I also discovered what appear to be some gross in-efficiencies in ProDOS. In the

course of even simple CATALOGs, LOADs, and SAVEs the same blocks are read into the same buffers over and over, at times when it would appear to be totally unnecessary. If there was some mechanism inside MLI to keep track of the fact that a complete un-spoiled copy of a particular block was already in RAM, it could save a lot of time. On the other hand, it could be that the current approach is safer. I think it is a potentially fruitful area for further investigation. Any takers?

```
                   *SAVE PRODOS.SNOOPER
                   *-------------------------------
                          .OR $300
                          .OP 65C02     (If you have one)
                   *-------------------------------
06-                SLOT   .EQ 6
02-                DRIVES .EQ 2
                   *-------------------------------
0800-              BUFFER .EQ $800
                   *-------------------------------
0300- 4C 06 03     A300   JMP INSTALL.SNOOPER
0303- 4C 55 03     A303   JMP DISPLAY
                   *-------------------------------
                   INSTALL.SNOOPER
0306- A2 01               LDX #1
0308- BD 1C BF     .1     LDA 2*SLOT+$BF10,X
030B- 48                  PHA          SAVE CURRENT DRIVER ADDRESS
030C- BD 23 03            LDA DRIVER,X INSTALL NEW DRIVER ADDRESS
030F- 9D 1C BF            STA 2*SLOT+$BF10,X
                     .DO DRIVES=2
0312- 9D 2C BF            STA 2*SLOT+$BF20,X
                     .FIN
0315- 68                  PLA          REMEMBER OLD DRIVER
0316- 9D 23 03            STA DRIVER,X
0319- BD 25 03            LDA BUFFER.ADDR,X
031C- 9D 4A 03            STA A+1,X
031F- CA                  DEX
0320- 10 E6               BPL .1       NOW THE OTHER BYTE
0322- 60                  RTS
                   *-------------------------------
0323- 27 03        DRIVER      .DA SNOOPER
0325- 00 08        BUFFER.ADDR .DA BUFFER
                   *-------------------------------
                   SNOOPER
0327- 48                  PHA
0328- 5A                  PHY          (If no 65C02 use TYA, PHA)
0329- DA                  PHX          (If no 65C02 use TXA, PHA)
032A- BA                  TSX
032B- BD 04 01            LDA $104,X   LO-BYTE OF RETURN ADDR
032E- 20 49 03            JSR STORE.BYTE
0331- BD 05 01            LDA $105,X   HI-BYTE OF RETURN ADDR
0334- 20 49 03            JSR STORE.BYTE
0337- A2 00               LDX #0       $42...47
0339- B5 42        .1     LDA $42,X    WHICH ARE THE PARAMETERS
033B- 20 49 03            JSR STORE.BYTE    FOR THE CALL
033E- E8                  INX
033F- E0 06               CPX #6
0341- 90 F6               BCC .1
0343- FA                  PLX          (If no 65C02 use PLA, TAX)
0344- 7A                  PLY          (If no 65C02 use PLA, TAY)
0345- 68                  PLA
0346- 6C 23 03            JMP (DRIVER) CONTINUE IN DRIVER
                   *-------------------------------
                   STORE.BYTE
0349- 8D 00 08     A      STA BUFFER   THIS ADDRESS IS MODIFIED
034C- EE 4A 03            INC A+1      BUMP PNTR TO NEXT ADDRESS
034F- D0 03               BNE .1
0351- EE 4B 03            INC A+2
0354- 60           .1     RTS
                   *-------------------------------
FDED-              COUT   .EQ $FDED
FD8E-              CROUT  .EQ $FD8E
FDDA-              PRBYTE .EQ $FDDA
00-                PNTR   .EQ $00,01
                   *-------------------------------
```

```
                  DISPLAY
0355- A9 00               LDA #BUFFER  SET UP PNTR INTO BUFFER
0357- 85 00               STA PNTR
0359- A9 08               LDA /BUFFER
035B- 85 01               STA PNTR+1
                  *---CHECK IF FINISHED-----------
035D- A5 00       .1      LDA PNTR
035F- CD 4A 03            CMP A+1
0362- A5 01               LDA PNTR+1
0364- ED 4B 03            SBC A+2
0367- 90 01               BCC .2
0369- 60                  RTS
                  *---DISPLAY NEXT 8 BYTES---------
036A- A0 01       .2      LDY #1
036C- 20 94 03            JSR WORD     DISPLAY RETURN ADDRESS
036F- A9 BA               LDA #":"     "XXXX:"
0371- 20 ED FD            JSR COUT
0374- 20 A1 03            JSR BYTE     DISPLAY ($42)=OPCODE
0377- 20 A1 03            JSR BYTE     DISPLAY ($43)=UNIT NUMBER
037A- C8                  INY
037B- 20 94 03            JSR WORD     DISPLAY ($44,45)=BUFFER ADDR
037E- 20 A6 03            JSR DOT
0381- 20 94 03            JSR WORD     DISPLAY ($46,47)=BLOCK NUMBER
0384- 20 8E FD            JSR CROUT    CARRIAGE RETURN
0387- A5 00               LDA PNTR     ADVANCE PNTR TO NEXT
0389- 18                  CLC                 GROUP OF 8 BYTES
038A- 69 08               ADC #8
038C- 85 00               STA PNTR
038E- 90 CD               BCC .1
0390- E6 01               INC PNTR+1
0392- D0 C9               BNE .1       ...ALWAYS
                  *-------------------------------
0394- B1 00       WORD    LDA (PNTR),Y        DISPLAY HI-BYTE
0396- 20 DA FD            JSR PRBYTE
0399- 88                  DEY                 DISPLAY LO-BYTE
039A- B1 00               LDA (PNTR),Y
039C- C8                  INY
039D- C8                  INY                 ADVANCE INDEX
039E- 4C DA FD            JMP PRBYTE
                  *-------------------------------
03A1- B1 00       BYTE    LDA (PNTR),Y        DISPLAY BYTE
03A3- 20 DA FD            JSR PRBYTE
03A6- A9 AE       DOT     LDA #"."            PRINT "."
03A8- C8                  INY                 ADVANCE INDEX
03A9- 4C ED FD            JMP COUT
                  *-------------------------------
```

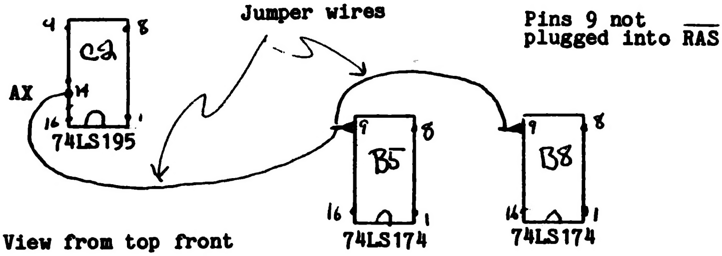
A Different Patch for 65C02 & Old Apples...William O'Ryan Jr.

Since my earlier letter (Jun 84) on the 65C02 and the Apple II+ I was interested and gratified to read Andrew Jackson's (Dec 84) and Jim Sather's (Mar 85) letters on the same subject. However, two things began to worry me. First, the smallness of the time gain in the F257 chips (around 7 nanoseconds, I understand). That did not seem enough to be very reliable. Second, a friend in town has an Apple whose speed was not sufficiently improved to allow the 65C02 to work (although there was some noticeable improvement).

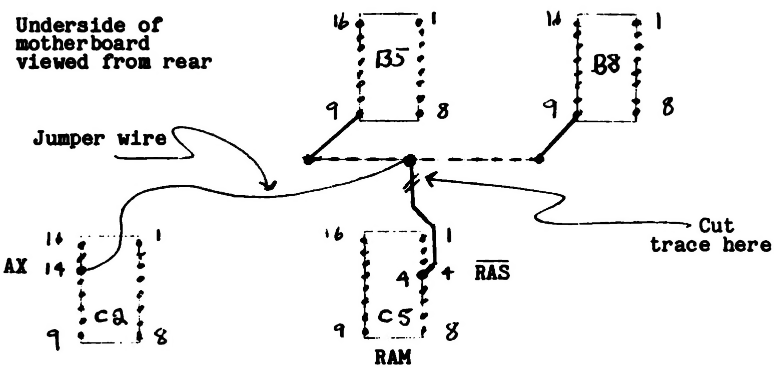
After reading the first few chapters of Jim Sather's book, "Understanding the Apple II", I was able to come up with a new solution. As I figure it, this new solution yields an improvement of around 70 nanoseconds, more than enough. Simply put, just replace the -RAS line inputs to the 74LS174 chips at B5 and B8 with AX. AX rises 70 nsec earlier than -RAS, enabling those chips to latch RAM output 70 nsec earlier. It is a simple patch and may be done either with or without altering the motherboard.

I tried it first without altering my motherboard, on a Rev 44-1 Apple using 200 nsec 16K RAM chips. I was surprised to see it work, as I had expected that 200 nsec RAM chips would be too slow for the patch. (I haven't tried it yet with 250 nsec RAM chips.) Actually, this particular Apple did not need any speed-up -- the 65C02 was already working in it.

To do this patch: remove the chips at B5 and B8; seat an extra socket under each of them; pin 9 on these sockets should be bent out so they do not go into the motherboard sockets; remove the chip at C2 and put an extra socket under it; connect a wire from pin 14 of the C2 socket to the bent out pins 9 of B5 and B8. Pin 14 of the 74S195 at C2 is a source of the AX signal; pin 9 of B5 and B8 was previously connected to -RAS.



I have another Apple (Rev 4) which has 24 150 nsec 64K RAM chips (using the Cramapple mod). This Apple already had F257's in it with a 65C02. I put the old LS257's back in, and sure enough the 65C02 began to stumble. Then I removed the motherboard and on the underside cut the trace to -RAS and soldered in a jumper wire to pin 14 of C2. It worked perfectly!



Naturally those who try any of these patches do so at their own risk.

I must thank Jim Sather for his book; it was only by studying the timing diagrams in that book and staring at the circuit diagram published by Apple that I was able to do this. I hope some of the hardware types will be able to tell me if I have built a time bomb. I am also very interested to hear whether the problem with the 65802 is the same.

Of course if I want to look around in ProDOS the same curiosity certainly applies to DOS 3.3. The fact of the matter is, I started snooping in DOS first; nevertheless, the ProDOS article took precedence in these pages.

There are several nice places to patch a snooper into DOS 3.3. One is right at the beginning of RWTS, $BD00. This position is usually taken by hard disks, however. For example, Sider and Corvus use $BD00. I could skip down below $BD00, but Sider for one expects several bytes after $BD00 to be normal DOS code. Looking backward, $BD00 is normally called only from a subroutine which starts at $B7B5. This subroutine, in turn, is normally only called from $B090. Your own programs may call RWTS differently, but DOS itself almost always goes through $B090. (The exceptions are the reading and writing of the DOS image during boot or INITialization.)

Therefore...I patched my SNOOPER program in at $B090. The INSTALL.SNOOPER code in lines 1060-1160 is very similar to that in the ProDOS snooper. It swaps the address currently in my variable DRIVER with the address at $B091,2. Typing "$800G" will install SNOOPER, and typing it again will dis-install SNOOPER.

The DOS snooper prints out each line of information as it goes along, without storing the data. Each line contains the two most recent return address from the stack, so you can trace who is calling RWTS. I also print out the RWTS command, the track and sector, and the buffer address.

Here is an example of the printout, in this case during a SAVE operation:

```
:LOAD S.RWTS.SNOOPER
:ASM                         Assembler SNOOPER

0000 ERRORS IN ASSEMBLY
:$800G                       install SNOOPER
:SAVE S RWTS.SNOOPER         sample DOS command
AB24.AD45.01.11.00.B3BB      read VTOC
AB45.B1E6 01.11.0F.B4BB      read Catalog sector
A6AA.AB24.01.1F.0F.9700      T/S list
C3E9.ACDD.01.1F.0E.9600      read 1st data sector
ACDD.B0C8.02.1F.0E.9600      write 1st data sector
D349.ACDD.01.1F.0D.9600      read 2nd data sector
ACDD.B0C8.02.1F.0D.9600      write 2nd data sector
D328.ACDD.01.1F.0C.9600      read 3rd data sector
ACDD.B0C8.02.1F.0C.9600      write 3rd data sector
D352.ACDD.01.1F.0B.9600      read 4th data sector
ACDD.B0C8.02.1F.0B.9600      write 4th data sector
A2F8.A6AA.01.11.00.B3BB      read VTOC
A6AA.AC1E.01 11.0F.B4BB      read catalog sector
A2F8.A6AA.02.11.0F.B4BB      write catalog sector
AD1A.AB45.01.11.00.B3BB      read VTOC
AB45.B1E6.01.11.0F.B4BB      read catalog sector
A6AA.AD1A.01.1F.0F.9700      read T/S list
```

```
A6AA.AD1D.01.1F.0E.9600   read 4 data sectors
A6AA.AD1D.01.1F.0D.9600     to VERIFY the file
A6AA.AD1D.01.1F.0C.9600
A6AA.AD1D.01.1F.0B.9600
:$800G                     dis-install SNOOPER
```

```
                  1000 *SAVE S.RWTS.SNOOPER
                  1010 *--------------------------------
FDDA-             1020 PRBYTE .EQ $FDDA
FD8E-             1030 CROUT  .EQ $FD8E
FDED-             1040 COUT   .EQ $FDED
                  1050 *--------------------------------
                  1060 INSTALL.SNOOPER
0800- A2 01       1070        LDX #1
0802- BD 14 08    1080 .1     LDA DRIVER,X
0805- 48          1090        PHA
0806- BD 91 B0    1100        LDA $B091,X
0809- 9D 14 08    1110        STA DRIVER,X
080C- 68          1120        PLA
080D- 9D 91 B0    1130        STA $B091,X
0810- CA          1140        DEX
0811- 10 EF       1150        BPL .1
0813- 60          1160        RTS
                  1170 *--------------------------------
0814- 16 08       1180 DRIVER .DA SNOOPER  MODIFIED DURING OPERATION
                  1190 *--------------------------------
                  1200 SNOOPER
0816- AD 78 07    1210        LDA $778
0819- 8D 74 08    1220        STA SAVE778
081C- AD F8 07    1230        LDA $7F8
081F- 8D 75 08    1240        STA SAVE7F8
                  1250 *--------------------------------
0822- BA          1260        TSX
0823- 20 8E FD    1270        JSR CROUT
0826- 20 5F 08    1280        JSR PRADDR   PRINT RETURN ADDR FROM STACK
0829- 20 5F 08    1290        JSR PRADDR   AND ANOTHER ONE
                  1300 *--------------------------------
082C- AD F4 B7    1310        LDA $B7F4    COMMAND
082F- 20 6A 08    1320        JSR BYTE
0832- AD EC B7    1330        LDA $B7EC    TRACK
0835- 20 6A 08    1340        JSR BYTE
0838- AD ED B7    1350        LDA $B7ED    SECTOR
083B- 20 6A 08    1360        JSR BYTE
083E- AD F1 B7    1370        LDA $B7F1    BUFFER ADDRESS
0841- 20 DA FD    1380        JSR PRBYTE
0844- AD F0 B7    1390        LDA $B7F0
0847- 20 DA FD    1400        JSR PRBYTE
                  1410 *--------------------------------
084A- AD 74 08    1420        LDA SAVE778
084D- 8D 78 07    1430        STA $778
0850- AD 75 08    1440        LDA SAVE7F8
0853- 8D F8 07    1450        STA $7F8
0856- AD C2 AA    1460        LDA $AAC2
0859- AC C1 AA    1470        LDY $AAC1
085C- 6C 14 08    1480        JMP (DRIVER)
                  1490 *--------------------------------
                  1500 PRADDR
085F- BD 08 01    1510        LDA $108,X
0862- 20 DA FD    1520        JSR PRBYTE
0865- BD 07 01    1530        LDA $107,X
0868- CA          1540        DEX          SET UP FOR NEXT ADDRESS
0869- CA          1550        DEX
086A- 20 DA FD    1560 BYTE   JSR PRBYTE
086D- A9 AE       1570        LDA #"."
086F- 4C ED FD    1580        JMP COUT
                  1590 *--------------------------------
0872-             1600 SAVEX  .BS 1
0873-             1610 SAVEY  .BS 1
0874-             1620 SAVE778 .BS 1
0875-             1630 SAVE7F8 .BS 1
                  1640 *--------------------------------
```

Feedback about the latest Sieve............Peter J. McInerney

So the Sieve lives! Bob's article last month misses some of the facts, however. He states that my improved 68000 version on my 12.5 MHz DTACK Grounded board ran in .4 seconds; the actual time was .33 seconds. This is proportional to the .49 seconds claimed in the later Byte article for an 8 MHz 68000. My DTACK Grounded board uses 120 nanosecond static RAM and runs at a full 12.5 MHz speed (DTACK grounded means that the processor CANNOT wait for memory).

Hal Hardenburgh (editor of the now sadly no more DTACK newsletter and no slouch when it comes to assembly programming on the 68000) produced his own version of the original algorithm, essentially hand-compiled BASIC since that was what he wanted to compare to, and that ran in 1.29 secs for 10 iterations on a 10MHz board.

My faster 68000 sieve was my first 68000 program, so in light of my now more extended experience I tried to tighten it up even further. The result runs in .28 seconds for ten iterations on my DTACK board, and .72 seconds on a Macintosh. The main speed improvement comes from loading two extra registers for comparisons rather than doing CMPI's. The use of MOVEM for clearing the array was pointed out to me by Hal Hardenburgh and accounts for about .02 secs saved, at the expense of a large amount of elegance (oh well, what price aesthetics?).

In trying to guess the comparisons of the 65816 systems of the future with existing 68000 systems, two questions come to mind. First, if 6 or 8 MHz 65816s become available in quantity, how fast will the memory have to be to keep up? The 68000 can automatically adjust for slower memories, but is this true of the 65816? Second, and more importantly, is the question of memory addressing.

I wrote a version of the sieve that sifts the first 262143 integers. This took 13.5 seconds for 10 iterations on a Macintosh (this should equate to 5.3 seconds on my DTACK board, but I don't have enough memory to test it.) The program is only minimally different from the original (some constants changed and some address modes changed from word to long.)

How about writing a 65816 program to handle this large of an array? How much extra baggage is required to test page boundaries, move base addresses, etc? My point is that the restriction of 64K banks can really hurt in accessing large data arrays. Memory is getting cheaper all the time, so using more bytes for a 68000 program may well be no penalty, compared with the extra difficulty of writing 65816 code to handle large amounts of data.

```
                  ;SAVE"FastestSieve.asm"
                  *--------------------------------
                  * 68000 Sieve Macintosh version October 1985
                  *--------------------------------
                  * This sieves the first 16383 integers ( $4000-1)
                  *--------------------------------
                  * This code is non relocatable and is run from
                  * the MacASM environment
                  *--------------------------------
                  *--------------------------------
0007A700:         SCREEN   EQU      $07A700
                  *--------------------------------
                           ORG      $15000
                  SysBeep  MACRO
                           HEX      A9C8
                           ENDM
                  *--------------------------------
00015000: 7002    START    MOVEQ    #2,D0
00015002: 3F00             MOVE     D0,-(A7)
00015004:                  SysBeep
00015004: A9C8   >          HEX      A9C8
00015006: 48E7 0004        MOVEM.L A5,-(A7)       Save for Macintosh
0001500A: 3C3C 03E7        MOVE     #1000-1,D6    Do 1000 times
                  *--------------------------------
0001500E: 33C6 0001
00015012: 50C0    .1       MOVE     D6,COUNT      Temporary save
00015014: 207C 0007
00015018: C718             MOVE.L   #SCREEN+$2000+$18,A0 Top of array plus
0001501A: 700C             MOVEQ    #12,D0        Count           overshoot
                  *--------------------------------
                  * Clear alll free registers
                  *--------------------------------
0001501C: 7200             MOVEQ    #0,D1
0001501E: 7400             MOVEQ    #0,D2
00015020: 7600             MOVEQ    #0,D3
00015022: 7800             MOVEQ    #0,D4
00015024: 7A00             MOVEQ    #0,D5
00015026: 7C00             MOVEQ    #0,D6
00015028: 7E00             MOVEQ    #0,D7
0001502A: 2241             MOVE.L   D1,A1
0001502C: 2441             MOVE.L   D1,A2
0001502E: 2641             MOVE.L   D1,A3
00015030: 2841             MOVE.L   D1,A4
00015032: 2A41             MOVE.L   D1,A5
00015034: 2C41             MOVE.L   D1,A6
                  *--------------------------------
                  * Clear array using register save
                  *--------------------------------
00015036: 48E0 7F7E .2     MOVEM.L D1-D7/A1-A6,-(A0)
0001503A: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
0001503E: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
00015042: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
00015046: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
0001504A: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
0001504E: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
00015052: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
00015056: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
0001505A: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
0001505E: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
00015062: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
00015066: 51C8 FFCE        DBF      D0,.2
                  *--------------------------------
                  * Finish array clear.  Overshoots by $18
                  *--------------------------------
0001506A: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
0001506E: 48E0 7F7E        MOVEM.L D1-D7/A1-A6,-(A0)
                  *--------------------------------
                  * Now do sieve
                  *--------------------------------
00015072: 7003             MOVEQ    #3,D0         Start with 3
00015074: 7804             MOVEQ    #4,D4         Corresponds to 9
00015076: 7404             MOVEQ    #4,D2         Difference
00015078: 76FF             MOVEQ    #$FF,D3       Used to knock out
0001507A: 207C 0007
0001507E: A701             MOVE.L   #SCREEN+1,A0  Position of 3
00015080: 227C 0007
00015084: A700             MOVE.L   #SCREEN,A1    Array start
00015086: 2E3C 0000
0001508A: 2000             MOVE.L   #$2000,D7     Set up for comparisons
```

```
0001508C: 727F               MOVEQ   #127,D1
0001508E: 6004               BRA.S   .4
                   *--------------------------------
00015090: 5842     .3        ADDQ    #4,D2         Update difference
00015092: D842               ADD      D2,D4        Update square pointer
00015094: 4A18     .4        TST.B   (A0)+         Knocked out yet?
00015096: 660C               BNE.S   .6            Yes
                   *--------------------------------
00015098: 3A04               MOVE    D4,D5         Get latest square
0001509A: 1383 5000 .5       MOVE.B  D3,0(A1,D5)   Knock out from here
0001509E: DA40               ADD     D0,D5         Ignore even multiples
000150A0: BA47               CMP     D7,D5
000150A2: 63F6               BLS     .5
                   *--------------------------------
000150A4: 5440     .6        ADDQ    #2,D0         Get next odd number
000150A6: B041               CMP     D1,D0         Got to sqrt $4000-1 yet?
000150A8: 63E6               BLS     .3            No
                   *--------------------------------
000150AA: 3C39 0001
000150AE: 50C0               MOVE    COUNT,D6      Recover loop count
000150B0: 51CE FF5C          DBF     D6,.1         Loop again
                   *--------------------------------
000150B4: 4CDF 2000          MOVEM.L (A7)+,A5      Restore for Macintosh
000150B8: 7002               MOVEQ   #2,D0
000150BA: 3F00               MOVE    D0,-(A7)
000150BC:                    SysBeep
000150BC: A9C8                HEX      A9C8
000150BE: 4E75               RTS
                   *--------------------------------
000150C0:          COUNT     DEFS    2
                             END
```

Paint Yourself into the Corner....................Adam Levin

I think I have come up with an interesting puzzle. Pretend that your Apple has only 48K of RAM: no ROM, no soft switches, no memory cards, just 48152 bytes of contiguous RAM from $0000 through $BFFF. Now, write a program which will store one number (of your choosing) into each and every one of these 49152 locations. The stumper here is creating a program which can overwrite itself completely, and which will not go running off through the I/O area causing disks to spin, etc.

There are certain limitations to actually implementing this on an Apple. When you hit <RESET> to examine the contents of memory after running your program, memory will be changed before you can look at it. It is unavoidable that page zero, the stack, and text screen memory will all get disrupted as soon as <RESET> is pressed. You still need to include these areas in your program, but you just will not be able to check them.

You will have to figure out some way of stopping the program before it runs off into the $Cxxx space. I decided to accept this limitation by allowing three bytes at $BFFD-F to contain a JMP instruction, not stuffing my favorite number in them. So my solution actually only stuffs my number into $0000-$BFFC.

Bob Sander-Cederlof has a solution that stuffs the same number in every byte from $0000 through $BFFF, but depends on two locations in the I/O area to stop the program from rampaging around $Cxxx space.

Try your hand at this puzzle! Next month we'll show some of the best solutions.

INDEX

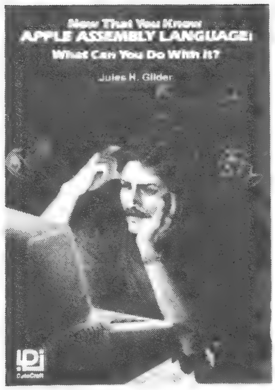
Apple Assembly Line, Volume 5
October, 1984 through September, 1985

AAAA

BBBB

CCCC

BECOME AN ASSEMBLY LANGUAGE PROGRAMMING WHIZ



You've spent a lot of time learning Apple assembly language and finally know the difference between BEQ and BCS. Now it's time to put your new-found knowledge to work. Time to throw away your Applesoft programming manual and write programs that make your Apple work like a super-charged, super-fast computer. Time to graduate from the Applesoft BASIC used by beginners, to the 6502 assembly language used by professionals.

To help make this transition, you need an experienced programmer to guide you. You need to develop a library of subroutines that make programming in assembly language as easy as programming in BASIC. You need to learn all the tricks that take experienced assembly language programmers years to acquire. Most important of all, you need the book, *"Now that You Know Apple Assembly Language: What Can You Do With It?"* because it contains all this information *and more.*

It shows you how, step-by-step

"Now That You Know Apple Assembly Language: What Can You Do With It?" will take you step-by-step through the assembly language programming experience. You'll delve into the mysteries of the 6502 stack and learn how to use it to increase the power and versatility of your programs. You'll also learn how to use the Apple's built-in routines to minimize the amount of coding you must do.

Control the output and the input

Frequently it's desirable to gain total control of the computer's output. This book shows you how to *steal control away from the Apple's normal output routines and redirect it to your own program.* Thus if you wanted, you could see the normally invisible control characters, display text on your screen as black on white instead of the normal white on black, format text sent to a printer into pages and much more.

Expand the power of your Apple by *stealing control away from the normal input routines.* Do things like adding a screen print capability, or *convert part of the normal keyboard into a numeric keypad.* It's even possible to *produce self-modifying programs* by EXECing in commands from RAM instead of from the disk drive. Think about the possibilities that offers for protecting your programs. When you want to go back to Applesoft programming, *you'll be able to do it faster with the aid of Applesoft Shorthand,* an assembly language program that types in one or more Applesoft commands at the press of a key, or use another program in the book to *automatically count the number of lines in your Applesoft program.*

With this book you'll also learn about *generating tones and how to figure out the frequency, producing sound effects, teaching your Apple to send Morse code, restoring accidentally erased Applesoft programs, adding new commands to Applesoft and running two Applesoft programs in memory together,* to name a few.

Everything is explained

Unlike other books that merely consist of a collection of programs, this one explains what's happening, where and why. You get detailed descriptions of how the programs work and detailed program listings with virtually every line of code explained. Nothing is left to chance or misinterpretation.

Order now, get 2 FREE gifts

The book costs only $19.95 plus $2 for shipping and handling. Order now and you'll also get a *FREE Programmer's Number Conversion System* that makes it easy to convert between binary, hexadecimal and decimal numbers. No calculators are required. You'll convert numbers almost instantly and wonder how you ever got along without it.

As an extra bonus for prompt ordering, you'll receive a *FREE coupon worth $5 off* the price of a disk with all the assembled programs on it or a disk that contains the source code. These disks normally sell for $15 each. We're offering these FREE gifts for a limited time only, so hurry! *Order today!*

Money-back guarantee*

We're so confident that you'll find this book invaluable and want it in your library, that we're offering a 10-day, no-questions-asked, money-back guarantee. Order the book. Read it and try the programs for ten days. At the end of ten days if you don't think it's worth every penny you paid for it, just send it back in resalable condition and we'll refund your money immediately, no questions asked.

Redlig Systems, Inc., Dept. A 978
2068—79th St., Brooklyn, NY 11214

Please rush me ____ copies of **"Now That You Know Apple Assembly Language: What Can You Do With It?"** at $19.95 each plus $2 shipping and handling. I understand that if I am not delighted with the book I may return it within 10 days for a prompt and courteous refund. In any case, the Programmer's Number Conversion System and $5 coupon are mine to keep.

[ ] Enclosed is my check for $ ________

Please charge my credit card:
[ ] American Express [ ] MasterCard [ ] Visa

Card No. ______________ Exp. ________

Signature ______________________________

Name ______________________________

Address ______________________________

City ____________ State ____ Zip ________

*NOTE: Shipping and handling fees are not refundable.

Multiple Column Dis-Assembly.......................Adam Levin

When I'm writing and debugging a program, I always use a lot of printer paper as I list and re-list version after version of my creation. Using the Apple monitor's 'L' command wastes a lot of that paper, too. Since each disassembled line takes at most 36 characters, I end up wasting half of each page!. I know I could feed the paper through a second time with the right hand side now on the left, but the left hand listing isn't always the same length as the right, so I end up with listings that span several separate lengths of paper I've written a program to solve this dilemma (as if you hadn't guessed!), and I call it PolyCol.

PolyCol will be of use no matter what type of printer you have: daisywheel printer and 80-column video card owners will get two columns per page (screen), 80-column dot matrix owners can get up to four columns per page by using compressed printing, and those with wider carriages can get even more! In addition, by compressing the print size vertically as well, it is possible to get a disassembly of all the ROMs in the Apple onto only 16 pages! (It's also possible to go blind trying to read it!)

Note that rather than creating all the text in memory, and then dumping an entire page at once, PolyCol calculates which opcode to disassemble where, 'on-the-fly'. You might think that this would slow things down appreciably; but in fact unless you require tens of columns, the listing is done relatively quickly.

As you will see from the listing, seven zero-page locations are used to hold the parameters which the user must specify. You must store the starting and ending addresses of the area to be dis-assembled into locations \$00-03. Locations \$04-07 control the number of lines per page and columns per line, as well as several other features. Here are some examples to show what you can do with different parameter settings:

\$04	\$05	\$06	
\$01	\$14	\$FE	- Standard monitor 'L' listing. Press any key to see the next page.
\$02	\$36	\$FF	- Two column, 54 line page with a form feed in between pages
\$04	\$4C	\$0C	- Four column, 76 line page with 12 spaces between pages. Don't forget to set elite typeface and compressed print.
\$04	\$70	\$FF	- Four column, 112 lines per page! To do this I had to use compressed elite super- script, with a line spacing of 1/12th in.

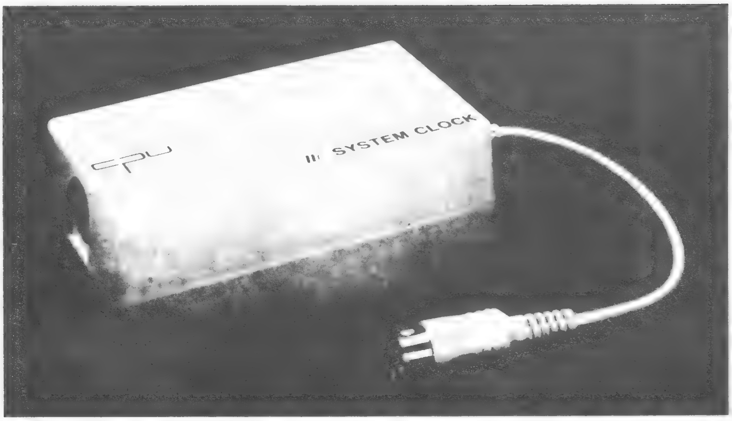
You can add just a little code to POLYCOL to set it up as a control-Y command. Then you could set the starting and ending

addresses as in normal monitor commands. The other four parameters could also be specified in the control-Y command format, if you really get serious about modifications.

```
                1000 *SAVE S.POLYCOL
                1010 *--------------------------------
                1020 *  PolyCol
                1030 *  Produces multi-column Apple monitor dis-assemblies.
                1040 *  Copyright (c) 1986 Adam Levin
                1050 *--------------------------------
                1060        .OR $800
                1070 *---User parameters-------------
00-             1080 STRTL  .EQ $00       Starting address
01-             1090 STRTH  .EQ $01
02-             1100 ENDL   .EQ $02       Ending address
03-             1110 ENDH   .EQ $03
04-             1120 NCPP   .EQ $04       # Columns per page
                1130 *                      (0 <= NCPP <= FF)
                1140 *                      (each column takes 34 chars.)
05-             1150 NLPP   .EQ $05       # Lines printed per page
                1160 *                      (0 <= NLPP <= FF)
06-             1170 NSKP   .EQ $06       # Blank lines between pages
                1180 *                      (0 <= NSKP <= FF)
                1190 *                      (FF = Form feed)
                1200 *                      (FE = pause between pages)
07-             1210 SLOT   .EQ $07       Slot # to direct output to
                1220 *                      (0 <= SLOT <= 7)
                1230 *                      (0 = use currently active device)
                1240 *---Program variables-----------
08-             1250 BRUNFX .EQ $08       Holds the DOS stack pointer
09-             1260 TOFARL .EQ $09       Adrs of 1st opcode in col 2;
0A-             1270 TOFARH .EQ $0A       1st column ends just before it.
0B-             1280 TCSWL  .EQ $0B       Holds the 'other' CSWL address
0C-             1290 TCSWH  .EQ $0C
0D-             1300 COLCNT .EQ $0D       Current column
0E-             1310 TEMPL  .EQ $0E       Temporary storage
0F-             1320 TEMPH  .EQ $0F           "         "
                1330 *---Monitor variables------------
2E-             1340 FORMAT .EQ $2E       Holds addressing mode code
36-             1350 CSWL   .EQ $36       Character Output SWitch Low address
37-             1360 CSWH   .EQ $37           "      "      "   High    "
3A-             1370 PCL    .EQ $3A       Adrs of opcode currently being
3B-             1380 PCH    .EQ $3B       dis-assembled.
AA59-           1390 STKPTR .EQ $AA59     DOS 3.3 stack pointer save loc't'n
C000-           1400 KBD    .EQ $C000     Keyboard
C010-           1410 STROBE .EQ $C010     Clear keyboard strobe
                1420 *---Monitor ROM Subroutines------
F88C-           1430 INSDS2 .EQ $F88C     Formats each disassembly line
F8D3-           1440 INSTDSPA .EQ $F8D3   Print opcode & operand
F94A-           1450 PRBL2  .EQ $F94A     Prints (X-reg) many blank spaces
F953-           1460 PCADJ  .EQ $F953     Adjusts A,Y (PCL.H) after each line
FD0C-           1470 RDKEY  .EQ $FD0C     Get an input character
FD8E-           1480 CROUT  .EQ $FD8E     Print a <RETURN>
FD99-           1490 PRYX2A .EQ $FD99     Print 'adrs-'
FDED-           1500 COUT   .EQ $FDED     Print Acc as a character
                1510 *---Macro definitions-----------
                1520        .MA CMPD      Double byte CMP
                1530        LDA ]1        From the S-C
                1540        CMP ]2        MACRO LIBRARY file.
                1550        LDA ]1+1
                1560        SBC ]2+1
                1570        .EM
                1580 *
                1590        .MA MOVD      Double byte MOV
                1600        LDA ]1
                1610        STA ]2
                1620        LDA ]1+1
                1630        LDA ]2+1
                1640        .EM
                1650 *
                1660        .MA MSG       MESSAGE PRINT MACRO
                1670        LDX #]1
                1680        JSR PRINT.MESSAGE
                1690        .EM
                1700 *--------------------------------
```

```
                  1710 POLYCOL
0800- AD 59 AA 1720         LDA STKPTR   Save stack pointer now,
0803- 85 08    1730         STA BRUNFX   restore it at the end.
0805- A5 07    1740         LDA SLOT     Send the output to another device?
0807- F0 06    1750         BEQ .1       No.
0809- 09 C0    1760         ORA #$C0     Use $Cn00 (n=SLOT) so we can simulate a
080B- A2 00    1770         LDX #0       PR#n when we swap CSWL,H & TCSWL,H.
080D- F0 04    1780         BEQ .2       This creates a problem if SLOT <> 0 &
080F- A5 37    1790 .1      LDA CSWH     SLOT contains an 80-col card since PR#
0811- A6 36    1800         LDX CSWL     can activate card, but not de-activate.
0813- 85 0C    1810 .2      STA TCSWH    No harm done, but it can be confusing.
0815- 86 0B    1820         STX TCSWL
0817- 4C 87 08 1830         JMP PAUSE2   Start out by waiting for a keypress.
               1840 *--------------------------------
081A- A5 05    1850 STRT    LDA NLPP     'CALC' NLPP lines from STRTL.H.
081C- 85 0E    1860         STA TEMPL    Adrs of the opcode just after the last
081E- A9 00    1870         LDA #0       one in column one.  Store in TOFARL,H
0820- 85 0F    1880         STA TEMPH    to keep STRTL,H from going beyond it.
0822- 20 19 09 1890         JSR CALC
0825-          1900         >MOVD PCL,TOFARL
0825- A5 3A    0000>         LDA PCL
0827- 85 09    0000>         STA TOFARL
0829- A5 3B    0000>         LDA PCL+1
082B- A5 0A    0000>         LDA TOFARL+1
082D- A9 01    1910 COLM1   LDA #1       Always start in column one.
082F- 85 0D    1920         STA COLCNT   Set COLCNT to 1
0831-          1930         >CMPD ENDL,STRTL   Have we finished?
0831- A5 02    0000>         LDA ENDL       From the S-C
0833- C5 00    0000>         CMP STRTL        MACRO LIBRARY file.
0835- A5 03    0000>         LDA ENDL+1
0837- E5 01    0000>         SBC STRTL+1
0839- B0 11    1940         BCS NOESC    No, ENDL,H >= STRTL.H
083B- 20 8E FD 1950         JSR CROUT    Yes, purge last printed line.
083E- 20 56 09 1960 ESC     JSR SWAP     <ESC> brings you here, too.
0841-          1970         >MSG M.BYE   Print end message.
0841- A2 0C    0000>         LDX #M.BYE
0843- 20 6B 09 0000>         JSR PRINT.MESSAGE
0846- A5 08    1980         LDA BRUNFX   Restore the stack pointer
0848- 8D 59 AA 1990         STA STKPTR
084B- 60       2000         RTS          All done.
084C-          2010 NOESC   >CMPD STRTL,TOFARL  About to pass col 2?
084C- A5 00    0000>         LDA STRTL       From the S-C
084E- C5 09    0000>         CMP TOFARL        MACRO LIBRARY file.
0850- A5 01    0000>         LDA STRTL+1
0852- E5 0A    0000>         SBC TOFARL+1
0854- 90 3F    2020         BCC NULINE   No, so continue
0856- A6 04    2030         LDX NCPP     Yes, so find the new first
0858- 20 3F 09 2040         JSR MULT     line for the new first column.
085B- 20 19 09 2050         JSR CALC
085E-          2060         >MOVD PCL.STRTL
085E- A5 3A    0000>         LDA PCL
0860- 85 00    0000>         STA STRTL
0862- A5 3B    0000>         LDA PCL+1
0864- A5 01    0000>         LDA STRTL+1
0866- A6 06    2070 NUPAGE  LDX NSKP     Page breaks
0868- E0 FE    2080         CPX #$FE
086A- F0 15    2090         BEQ PAUSE    Pause
086C- B0 0B    2100         BCS FRMFD    Form feed
086E- E0 00    2110         CPX #0
0870- F0 A8    2120 .1      BEQ STRT     No break - solid listing
0872- 20 8E FD 2130         JSR CROUT    Yes, print NSKP lines
0875- CA       2140         DEX
0876- 4C 70 08 2150         JMP .1
               2160 *--------------------------------
0879- A9 8C    2170 FRMFD   LDA #$8C
087B- 20 ED FD 2180         JSR COUT
087E- 4C 1A 08 2190         JMP STRT
               2200 *--------------------------------
0881- 20 8E FD 2210 PAUSE   JSR CROUT    Print a <RETURN>
0884- 20 56 09 2220         JSR SWAP     Swap TCSWL,H & CSWL
0887-          2230 PAUSE2  >MSG M.PAUSE Print PAUSE msg
0887- A2 00    0000>         LDX #M.PAUSE
0889- 20 6B 09 0000>         JSR PRINT.MESSAGE
088C- 20 0C FD 2240         JSR RDKEY
088F- 20 56 09 2250         JSR SWAP     Swap back
0892- 4C 1A 08 2260         JMP STRT     Do it all again
```

NEW PRODUCT



IIc SYSTEM CLOCK

- Fully ProDos compatible
- Automatic time and date stamping
- Easy to use from BASIC
- Battery operated, uses 3 "AA" batteries (will last 1-2 years before simple replacement)
- Date has year, month, date and day of week
- Time has hours, minutes and seconds
- Will time and date stamp AppleWorks files
- Will display time and date on the AppleWorks screen
- Auto access from AppleWorks data-base (just use a time and date field)
- Pass through serial port - The IIc system clock can plug into either the modem or printer serial port, then modem or printer plugs into the clock
- No hassle 5 year warranty
- Only $79.00

"We Set the Standard"

214-241-6060 APPLIED ENGINEERING 9 AM - 11 PM

```
                2270 *-------------------------------
0895- 20 8E FD  2280 NULINE JSR CROUT     Print a <RETURN>
0898- AD 00 C0  2290        LDA KBD       A key might have been pressed
089B- 49 9B     2300        EOR #$9B      It might have been <ESC>
089D- D0 06     2310        BNE OFFSET    It wasn't; continue
089F- 2C 10 C0  2320        BIT STROBE    It was!  ESCape!
08A2- 4C 3E 08  2330        JMP ESC
08A5- A6 0D     2340 OFFSET LDX COLCNT    Compute which opcode to
08A7- 20 3F 09  2350        JSR MULT      Disassemble next.
08AA- 20 19 09  2360        JSR CALC
08AD-           2370        >CMPD ENDL,PCL  Is adrs be beyond ENDL,H?
08AD- A5 02     0000>        LDA ENDL        From the S-C
08AF- C5 3A     0000>        CMP PCL       MACRO LIBRARY file.
08B1- A5 03     0000>        LDA ENDL+1
08B3- E5 3B     0000>        SBC PCL+1
08B5- 90 4C     2380        BCC NEXTOP    Yes, don't bother with it
08B7- A6 3A     2390        LDX PCL       No, so disassemble it
08B9- A4 3B     2400        LDY PCH
08BB- 20 99 FD  2410        JSR PRYX2A    Print the opcode address
08BE- A2 01     2420        LDX #1
08C0- 20 4A F9  2430        JSR PRBL2     Print 1 blank.  Monitor puts three
                2440 *                    here, but if each column is no more
                2450 *                    than 34 chars long, can fit 4 columns
                2460 *                    onto a printer with 132 chars/line.
08C3- 20 8C F8  2470        JSR INSDS2    Format it
08C6- 20 D3 F8  2480        JSR INSTDSPA    Print it
08C9- A5 0D     2490        LDA COLCNT    If last column, don't pad.
08CB- C5 04     2500        CMP NCPP
08CD- F0 2C     2510        BEQ NXTCOL    It is, get out
08CF- A2 00     2520        LDX #0        Isn't, so pad with blanks so that each
                2530 *                    column takes exactly 34 characters.
08D1- 20 8C F8  2540        JSR INSDS2    Calculate the format code
08D4- A2 0A     2550        LDX #10       ASSUME 10 SPACES
08D6- A5 2E     2560        LDA FORMAT    Get it
08D8- F0 1E     2570        BEQ SPACE     1 byte code requires 10 spaces
08DA- A2 07     2580        LDX #7        ASSUME 7 SPACES
08DC- C9 81     2590        CMP #$81      Z-page
08DE- F0 18     2600        BEQ SPACE
08E0- CA        2610        DEX           ASSUME 6 SPACES
08E1- C9 21     2620        CMP #$21      Immediate
08E3- F0 13     2630        BEQ SPACE
08E5- CA        2640        DEX           ASSUME 5 SPACES
08E6- C9 82     2650        CMP #$82      Absolute
08E8- F0 0E     2660        BEQ SPACE     5 SPACES
08EA- C9 85     2670        CMP #$85      Zpage,Y
08EC- F0 0A     2680        BEQ SPACE     5 SPACES
08EE- C9 91     2690        CMP #$91      Zpage,X
08F0- F0 06     2700        BEQ SPACE     5 SPACES
08F2- C9 9D     2710        CMP #$9D      Relative
08F4- F0 02     2720        BEQ SPACE     5 SPACES
08F6- A2 03     2730        LDX #3        All others
08F8- 20 4A F9  2740 SPACE  JSR PRBL2     Print (X-reg) many blanks
08FB- E6 0D     2750 NXTCOL INC COLCNT    Go to next column
08FD- A5 04     2760        LDA NCPP
08FF- C5 0D     2770        CMP COLCNT    Have we gone too far?
0901- B0 A2     2780        BCS OFFSET    No, do OFFSET
0903- A9 01     2790 NEXTOP LDA #1        Jump over the line
0905- 85 0E     2800        STA TEMPL     just done.
0907- A9 00     2810        LDA #0
0909- 85 0F     2820        STA TEMPH
090B- 20 19 09  2830        JSR CALC
090E-           2840        >MOVD PCL.STRTL    Store it in STRTL,H
090E- A5 3A     0000>        LDA PCL
0910- 85 00     0000>        STA STRTL
0912- A5 3B     0000>        LDA PCL+1
0914- A5 01     0000>        LDA STRTL+1
0916- 4C 2D 08  2850        JMP COLM1     And do it all again
                2860 *-------------------------------
```

```
                2870 *  CALC returns the opcode adrs that is TEMPL,H
                2880 *       disassembled (!) lines from STRTL,H
                2890 *       It returns this address in PCL,H
0919-           2900 CALC   >MOVD STRTL,PCL   Put STRTL.H into PCL.H for INSDS1
0919- A5 00     0000>        LDA STRTL
091B- 85 3A     0000>        STA PCL
091D- A5 01     0000>        LDA STRTL+1
091F- A5 3B     0000>        LDA PCL+1
0921- A5 0E     2910 .1     LDA TEMPL      If TEMPL,H = 0 then done
0923- 05 0F     2920        ORA TEMPH
0925- F0 17     2930        BEQ .3
0927- A2 00     2940        LDX #0
0929- 20 8C F8  2950        JSR INSDS2     Get end of the next opcode & operand
092C- 20 53 F9  2960        JSR PCADJ      Get the new address from PCADJ
092F- 85 3A     2970        STA PCL        Store the resulting address in PCL,H
0931- 84 3B     2980        STY PCH
0933- A5 0E     2990        LDA TEMPL      DEC TEMPL,H - with help
0935- D0 02     3000        BNE .2         from the MACRO LIBRARY again!
0937- C6 0F     3010        DEC TEMPH
0939- C6 0E     3020 .2     DEC TEMPL
093B- B8        3030        CLV            Exit from top of loop, not here
093C- 50 E3     3040        BVC .1         Always taken
093E- 60        3050 .3     RTS
                3060 *--------------------------------
                3070 *  MULT returns (NLPP * n-1).  N is usually
                3080 *       COLCNT, and as such is usually a small
                3090 *       number (almost always smaller than NLPP).
                3100 *       So MULT simply adds NLPP to itself n times.
                3110 *       Returns with result in TEMPL,H
093F- A9 00     3120 MULT   LDA #0         Zero TEMPL,H
0941- 85 0E     3130        STA TEMPL
0943- 85 0F     3140        STA TEMPH
0945- 18        3150 .1     CLC
0946- CA        3160 .2     DEX            Exit loop from top, so call with n+1
0947- F0 0C     3170        BEQ .3         Anything times 0 equals 0
0949- A5 0E     3180        LDA TEMPL      Add NLPP to TEMPL.H
094B- 65 05     3190        ADC NLPP
094D- 85 0E     3200        STA TEMPL
094F- 90 F4     3210        BCC .1         ...NO CARRY, KEEP ADDING
0951- E6 0F     3220        INC TEMPH      ...CARRY
0953- B0 F0     3230        BCS .1         ...ALWAYS
0955- 60        3240 .3     RTS
                3250 *--------------------------------
0956- A5 36     3260 SWAP   LDA CSWL       Swap output device adrses.  They are
0958- A6 0B     3270        LDX TCSWL      the same if SLOT = 0, but swap anyway.
095A- 86 36     3280        STX CSWL
095C- 85 0B     3290        STA TCSWL
095E- A5 37     3300        LDA CSWH
0960- A6 0C     3310        LDX TCSWH
0962- 86 37     3320        STX CSWH
0964- 85 0C     3330        STA TCSWH
0966- 60        3340        RTS
                3350 *--------------------------------
0967- 20 ED FD  3360 PM.1   JSR COUT
096A- E8        3370        INX
                3380 PRINT.MESSAGE
096B- BD 71 09  3390        LDA MSGS,X
096E- 30 F7     3400        BMI PM.1
0970- 60        3410        RTS
                3420 *--------------------------------
                3430 MSGS
00-             3440 M.PAUSE    .EQ *-MSGS
0971- D0 D2 C5
0974- D3 D3 A0
0977- C1 A0 CB
097A- C5 D9 20  3450        .AT -'PRESS A KEY '
0C-             3460 M.BYE      .EQ *-MSGS
097D- AA AA AA
0980- A0 C5 CE
0983- C4 A0 CF
0986- C6 A0 CC
0989- C9 D3 D4
098C- C9 CE C7
098F- 20        3470        .AT -'*** END OF LISTING '
                3480 *--------------------------------
```

Apple Manuals from Addison-Wesley....................Bill Morgan

Those elusive Apple technical manuals are finally coming out of hiding! As we reported some months ago, Addison-Wesley is beginning to distribute Apple's manuals, and we can now supply them for you. The ones we have seen are at least as good as Apple's own editions, and in some cases better.

Here are the titles that we can order for you:

Applesoft Tutorial - $29.95, disk. Beginner's introduction to Applesoft, with a disk of examples.

Applesoft BASIC Programmer's Reference Manual - $22.95, 373+xxv pages. Complete reference manual for Applesoft, documenting all features with many examples.

BASIC Programming with ProDOS - $29.95, 264+xxix pages, disk Covers using ProDOS from BASIC, including command and file handling. The disk includes lots of examples, and the useful Applesoft Programmer's Assistant program, which includes RENUMBER, MERGE, AUTOmatic line numbering, REM deletion, variable cross reference, and other features.

And here are the ones that look most important, that we expect to keep in stock here at S-C:

Apple //e Technical Reference manual - $24.95, 409+xxxii pages. Here's Apple's documentation of all the internals of the //e, including I/O devices and firmware, memory organization, the System Monitor, peripheral-card programming, the Super Serial Card, and hardware implementation. The new edition includes all the new features of the Enhanced //e and a complete source listing of the ROMs. This book is essential for serious //e programming.

Apple //c Technical Reference Manual - $24.95. And here is the same detailed coverage of the //c, and more. Additional topics documented in this book are the built-in serial I/O ports, the mouse input, and interrupt handling. If you want to use these features of the //c, get this book.

ProDOS Technical Reference Manual- $29.95, 186+xvii pages, disk. This is the official book on ProDOS, covering files, MLI calls, System programs, interrupt handling, and more. The disk is the ProDOS Exerciser. which allows you to experiment with all of the MLI calls without writing special programs. This book completes a ProDOS programmer's reference shelf, along with Beneath Apple ProDOS, and Apple ProDOS: Advanced Features.

The //e manual was scheduled for July publication: we just received it and the ProDOS manual today. The //c manual is scheduled for November delivery: we'll accept orders and ship the book as soon as A-W comes through.

Many thanks to Apple and to Addison-Wesley for making these important documents so easily available.

Now That You Know Apple Assembly Language, What Can You Do With It?..................Review by Bill Morgan

Do you know the difference between LDA LABEL,X and LDA (LABEL),Y but wonder when to use which? Are you confused by the way PHA, PHA, RTS doesn't go home, but jumps somewhere else entirely? Do you know what the 6502 opcodes do, but still feel lost when it comes time to combine them into a program?

Jules Gilder, a long-time contributor to several of the Apple Magazines, has written a book just for you. He spends about 190 pages covering the intermediate level of assembly language programming in the Apple II computer. His programs are very well commented, and the accompanying text contains almost a line-by-line discussion of how and why each program works.

Gilder concentrates on the Apple-specific features of 6502 programming: input and output hooks, the internal speaker, and basic linkage to Applesoft. This combination should make this book especially appealing to those of you who have learned 6502 from a "generic" book and want to find out how to apply your new knowledge to your Apple II's.

Here is a summary of each chapter of Now That You Know...:

1) Before You Get Started -- This is an introduction to assemblers and their conventions.

2) Getting Information out of Your Computer -- This chapter covers simple output, including message printing and decimal number display.

3) Getting Information into Your Computer -- Here we get into reading keystrokes and lines, handling decimal input, and also menu control structures.

4) Stealing Control of the Output -- This one goes into taking over the output hook to do custom printer setup codes and drivers, output filtering, and formatting.

5) Stealing Control of the Input -- Learn how to grab the input hook to add a custom cursor, numeric keypad, an in-memory EXEC simulator, an Applesoft keyboard macro facility, and a lower-case input driver using the shift-key modification.

6) Using Sound in Your Programs -- How to use the Apple's built-in speaker to create a variety of sounds.

7) Learning to Use the Ampersand -- Here are techniques for hooking into the &-vector to do hexadecimal input and output in Applesoft, find a program line in memory, append two Applesoft programs, and revive a program lost by the NEW command.

8) Expanding Applesoft BASIC -- Now we can have computed GOTO, GOSUB and LIST, do double-byte PEEKs and POKEs, switch between two Applesoft programs sharing memory and variables, and add function keys to control output modes.

12 Good Reasons Why RAMWORKS™ Is The Best Expansion Card For Your IIe

1 APPLEWORKS MEMORY Even though Ramworks enhances and expands a VAST ARRAY of other programs, Appleworks is our claim to fame. A 64K Ramworks will ADD [illegible] of available desktop memory, a 128K Ramworks will ADD [illegible], a 256K Ramworks will ADD 182K and a 512K Ramworks will ADD [illegible], and a 1 meg Ramworks will give you nearly an 800K desktop. And it's all done automatically! When you plug in more memory chips into your Ramworks card, Appleworks will find them automatically. Ramworks also increases the maximum number of records from 1350 to 4300.

2 APPLEWORKS SPEED AND POWER Ramworks does more than just increase the desktop memory (as if that weren't enough). With Ramworks, Appleworks will be able to run up to 20 times faster. If you buy a 256K or larger Ramworks card, Appleworks will automatically load itself in Ramworks. This greatly increases the speed at which Appleworks operates by eliminating all that nasty time consuming disk access on Drive 1. These are but a few reasons why we say that Ramworks is Appleworks best friend.

3 EXPANDABILITY Ramworks was designed with the future in mind, as your needs increase, so can Ramworks. Clear instructions show you how to plug in more memory (up to 1 meg).

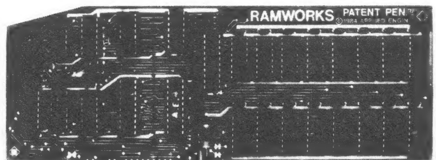
4 SPEED Today, as programs become more and more sophisticated, they inevitably become larger. And many of today's best selling programs (like Appleworks) won't fit in a 128K Apple, so many of these new larger programs continually go back to disk in search of more data. With Ramworks, you can have enough memory so that the entire program will be loaded into Ramworks' memory. This greatly increases the speed of software because your disk runs at 300 RPM, but Ramworks operates at the speed of light!

5 COLOR The same slot that's used for memory expansion is also the slot that's used for RGB color display, so all those lesser memory cards of yesterday make you decide in advance if you want RGB color. Only Ramworks lets you decide later to add RGB color. For only $129, an RGB option can be added to Ramworks to give you double high resolution color graphics and 80 column text. All with a razor sharp, vivid brilliance that's unsurpassed in the industry. The RGB option does not waste another valuable slot, but rather plugs into the back of Ramworks and attaches to any Apple compatible monitor. Remember, you can order the RGB option with your Ramworks or add it on at a later date.

6 COMPATIBILITY, OF THE SOFTWARE KIND Programs like Appleworks, Magic Office System, Flashcalc, The Spread Sheet, Diverse-A-Dos, Supercalc, Magicalc and many others automatically recognize all or most of Ramworks memory (512K is average). The simple fact is that Ramworks is compatible with more off the shelf software than any other RAM card. Ramworks is 100% compatible with ALL software written for the Apple 80 column and extended 80 column card. Additionally, Ramworks can emulate other RAM cards so software written for other cards will run without modification. Software written for RAMWORKS will not work on other cards. We can emulate others, but others can't emulate us.

7 COMPATIBILITY, OF THE HARDWARE KIND Unlike others, Ramworks is fully compatible with hardware add on's from other companies, like the Sider and Profile hard disks. And Ramworks was designed in accordance with the official expansion rules defined by Apple so you don't have to worry about compatibility problems. As you continue to expand and make your Apple more powerful with other expansion products from Applied Engineering, you'll appreciate how each product has extra features designed to work with Ramworks and other products to give you a total performance package that is more powerful than the sum of its parts.

8 IT SELLS THE MOST Popularity translates into great software support because software companies can't support all RAM cards, they can only support the ones their customers are likely to own. And software companies appreciate the fact that when they write software for Ramworks in the IIe,



they're also writing software for our memory expansion card for the IIc, Z-RAM. And our customer list reads like the Who's Who of Apple computing with just about every software company in the land buying one, including Apple Computer (in the hundreds), Rupert Lissner, and Steve Wozniak (we didn't give one to Mr. Wozniak just to use his name, 2 one meg Ramworks were paid for at full price).

9 IT'S FROM APPLIED ENGINEERING Unlike most of the competition, we only make accessories for Apple, so we'll never spend your money on IBM product research. Applied Engineering's years of experience and wide product line really pays off, and because of our high sales levels we buy most of our I.C. chips factory direct. So don't let our low prices fool you, they're caused by high volume production. That's why we can offer the most memory for the least money. Guaranteed!

10 IT'S GOT IT ALL

- ☑ Sharp 80 Column Text
- ☑ Double high resolution graphics (with or without RGB option)
- ☑ User Expandable to 1 Megabyte
- ☑ Can Use 64K or 256K RAMS in any combination
- ☑ Adds Memory to Appleworks
- ☑ Accelerates Appleworks
- ☑ 100% Compatibility with All IIe software
- ☑ RAM Disk software available, compatible with Applesoft, PRO-DOS, DOS 3.3, and PASCAL ($29)
- ☑ RAM Disk available for CP/M ($29). (This program is included with our CP/M card)
- ☑ Visicalc preboot available ($29)
- ☑ RGB option
- ☑ Takes only one slot
- ☑ 3 year no hassle warranty

11 THE PATENT OFFICE HAS ONE There are many advanced features on Ramworks, but two parts of the design are so advanced we applied for patents. One patent application deals with our ultra fast, ultra smooth 80 column screen display, and the other patent application deals with our ingenious way of dramatically reducing the power and heat of memory chips and improving reliability at the same time.

12 HERE TODAY, HERE TOMORROW In the seven years we've been making products for the Apple, we've seen a lot of companies come and go. Although nothing is forever, we're growing, expanding and we're profitable. And we are totally committed to Apple computing, which means you'll never run out of things to do with Ramworks. Or for that matter, reasons to buy one.

Ramworks™ with 64K	$179
Ramworks™ with 128K	$249
Ramworks™ with 256K	$299
Ramworks™ with 512K	$399
Ramworks™ with 1 MEG	$649
RGB Option (can be added later)	$129

Call (214) 241-6060
9 a.m. to 11 p.m. 7 days a week or send check or money order to:
Applied Engineering, P. O. Box 798, Carrollton, Texas 75006

MasterCard, Visa and C.O.D. welcome. No extra charge for credit cards. Texas Residents add 5⅛% sales tax. Add $10.00 if outside U.S.A.

APPLIED ENGINEERING
"We Set the Standard"

The only real weakness in this book is the complete lack of attention to the Apple's graphic display possibilities, and comparatively little coverage of dealing with DOS (and only one small appendix covering conversion to ProDOS.) I suppose Gilder regards these as more advanced topics. Hopefully he will see fit to focus on such subjects in a future book.

Gilder's company, Redlig Sytems, Inc., also has diskettes of all the programs in the book, in either source or object form.

We'll be carrying Now That You Know... for only $18 + shipping.

Apple Software Protection Digest

Gilder is also starting a newsletter on the subject of Apple software protection. This publication is devoted both to protecting your own programs and defeating the protection on others'. Here is part of Jules' description:

> Apple computer owners need a place where they can get more information about software protection. They need a forum where they can exchange ideas with others who face the same or similar problems. They need to know what software protection is, how it's implemented, what are the consequences of it, how it can be overcome if necessary and if there are any comparable unprotected alternatives to particular protected software packages.
>
> Apple Software Protection Digest will provide you with this information and more. It will show you new ways to protect, unprotect and backup your programs. It will teach you how to prevent others from accessing your programs and it will show you how to make them more difficult to copy. In addition, you'll learn how to overcome these and other protection schemes that are in use. You'll learn how to use the powerful, but complicated nibble copy programs. You'll also learn how to crack or remove protection entirely from many programs.

In the first issue he covers hiding Applesoft program lines (and finding them once they're hidden), making a machine language program automatically execute when BLOADed, protecting a disk by adding extra tracks and leaving some tracks unformatted, backing up The Print Shop, and he reviews the Copy II Plus nibble copier.

As a special offer for AAL subscribers, Gilder will give you a free copy of the first issue of Apple Software Protection Digest. Just send your name and address to Redlig Systems, Inc., 2068) 79th St., Brooklyn, NY, 11214. Be sure to mention that you are an AAL reader. The subscription rate is $24 for one year, or $42 for two years.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).